2014

TR-2014007: Real Polynomial Root-Finding by Means of Matrix and Polynomial Iterations

Victor Y. Pan

Real Polynomial Root-finding by Means of Matrix and Polynomial Iterations

Victor Y. Pan
Department of Mathematics and Computer Science
Lehman College of the City University of New York
Bronx, NY 10468 USA
Ph.D. Programs in Mathematics and Computer Science
The Graduate Center of the City University of New York
New York, NY 10036 USA
victor.pan@lehman.cuny.edu
http://comet.lehman.cuny.edu/vpan/

Abstract

Recently we proposed to extend the matrix sign classical iteration to the approximation of the real eigenvalues of a companion matrix of a polynomial and consequently to the approximation of its real roots. In our present paper we advance this approach further by combining it with the alternative square root iteration for polynomials and also show a variation using repeated squaring in polynomial algebra.

Keywords: Polynomials, Real roots, Companion matrix, Eigenvalues, Frobenius algebra, Matrix sign iteration, Square root iteration, Root squaring iteration

1 Introduction

Univariate polynomial root-finding is a classical and highly developed area but still an area of active research (see [29]– [32] and the bibliography therein). The problem is equivalent to approximating the eigenvalues of the companion matrix of the input polynomial, and it is tempting to employ the highly developed machinery of eigen-solving to polynomial root-finding. The papers [6], [45], [46], [48] studied the solution by means of the the Rayleigh Quotient iteration. This has a number of advantages. The iteration preserves matrix structure, allow us to approximate concurrently up to k eigenvalues (as long as k processors are available, which we would use with no date exchange among them), allow us to approximate multiple and clustered eigenvalues, and empirically converge fast, particularly near the eigenvalues. The QR algorithm, celebrated as a fast and very robust eigen-solver, has been also adjusted to polynomial root-finding in [8], [4], [7], [59], [3], [59], [1].

We use these two iterations for some auxiliary computations, but focus on a special and highly important case for which the cited algorithms are not particularly efficient. Namely in many applications, such as algebraic and geometric optimization, one seeks only r real roots of a degree n polynomial, which can have many more nonreal roots, that is $n \gg r$. In spite of this motivation, the fastest known algorithms supporting the record complexity estimates for the real root approximation are still the same as for the approximation of all complex roots [35]–[37], [40], [20],[52], [28], [57], [32].

Likewise the cited works exploiting matrix methods take no advantage when one seeks only real eigenvalues and roots. They compute the real eigenvalues by using order of n^2 arithmetic operations, the same as for all complex eigenvalues. Similarly MPSolve [5], [12] and Eigensolve [21], the current best packages of subroutines for polynomial root-finding, approximate the real roots of a polynomial about as fast and as slow as all its complex roots.

We explore and amend a distinct approach, studied in [53], [54], [24], [14], [10], and [41] and exploiting the matrix sign iteration, linked to the square root iteration for polynomials. These classical iterations are usually applied under the assumption that the roots of the input polynomial and the eigenvalues of its companion matrix are isolated from the imaginary axis [14], [10], [25]. On the contrary, by following and extending [48] and [46], we explore the application of this algorithm to the approximation of the eigenvalues that lie on that axis. Namely we reduce to this case the approximation of the real roots and eigenvalues simply by means of the rotation of the complex plane. As in [46] we approximate the real roots by applying the matrix sign iteration to the companion matrix and by exploiting its structure, but presently we accelerate the algorithms by combining them with the square root iteration for polynomials. Our resulting algorithms approximate all real roots by using order of $O(nr)$ arithmetic operations, up to logarithmic factors. For $r \ll n$ this is much less then order of n^2 in the cited algorithms.

Our techniques, particularly our interplay with matrix and polynomial computations to the benefit of both subjects, can be of independent interest (cf. [34], [9], [39]), as well as our exploitation of the complex plane geometry and various transforms of the variable. Even our simple recipe for real root-finding by means of combining the root radii algorithm with Newton's iteration in Remark 2.1 can be useful for a large class of inputs. In our concluding Section 5 we indicate some directions for further improvement of our real root-finders, in particular by exploiting the duality between matrix and polynomial computations more intensively, by using scaling and shifts of the variables and matrices and scaling of polynomials towards numerical stabilization of our algorithms and acceleration of their convergence, and by applying repeated squaring in polynomial algebra.

2 Basic Results

Hereafter "ops" stands for "arithmetic operations", "lc(p)" stands for "the leading coefficient of $p(x)$". $A(X,R,r) = \{x : r \le |x - X| \le R\}$, $D(X,r) = \{x : |x - X| \le r\}$, and $C(X,r) = \{x : |x - X| = r\}$ denote an annulus, a disc, and a circle on the complex plane, respectively. We write $||\sum_i v_i x^i||_q = (\sum_i |v_i|^q)^{1/q}$ for $q = 1, 2$ and $||\sum_i v_i x^i||_q = \max_i |v_i|$ for $q = \infty$. A function is in $\tilde{O}(f(b,n))$ if it is in $O(f(b,n))$ up to polylogarithmic factors in b and n. We assume a polynomial

$$p(x) = \sum_{i=0}^{n} p_i x^i = p_n \prod_{j=1}^{n} (x - x_j), \quad p_n \neq 0, \tag{2.1}$$

having real coefficients $p_0, \dots, p_n$, such that $||p_i||_\infty \le 2^\gamma$ for a fixed real γ, having r real roots $x_1, \dots, x_r$, and having $s = (n-r)/2$ pairs of complex conjugate roots $x_{r+1}, \dots, x_n$, where all roots are not necessarily distinct.

ROOT RADII APPROXIMATION

Theorem 2.1. *(Cf. [55], [32, Section 15.4].) Assume a polynomial $p(x)$ of (2.1) and two scalars $c > 0$ and d. Define the* root radii $r_j = |x_j|$ *for $j = 1, \dots, n$ and $r_1 \ge r_2 \ge \cdots \ge r_n$, so that all roots lie in the disc $D(0, r_1)$. Then approximations $\tilde{r}_j$ such that $\tilde{r}_j \le r_j \le (1 + c/n^d)\tilde{r}_j$ for $j = 1, \dots, n$ can be computed by using $O(n \log^2 n)$ ops.*

ROOT TRANSFORMS: SHIFT, SCALING, INVERSION, SQUARING.

We can invert all roots by reversing the polynomial, which involves no ops,
$p_{\rm rev}(x) = x^n p(1/x) = \sum_{i=0}^{n} p_i x^{n-i} = p_n \prod_{j=1}^{n} (1 - x x_j)$.
We can also shift and scale the roots at a low arithmetic cost.

Theorem 2.2. *(Cf. [39].) Given a polynomial $p(x)$ of (2.1) and two complex scalars a and b, one can compute the coefficients of the polynomial $q(y) = p(ay + b)$ by using $O(n \log n)$ ops. Only $2n - 1$ ops are needed if $b = 0$.*

By combining Theorems 2.1 and 2.2 we can move the roots of a polynomial into a fixed disc, e.g., $D(0,1) = \{x : |x| \le 1\}$.

Theorem 2.3. *The map $q(y) = (-1)^n p(\sqrt{x})p(-\sqrt{x})$ squares the roots of a polynomial $p(x)$ of (2.1), that is $q(y) = \prod_{j=1}^{n}(y - y_j)$ where $y_j = x_j^2$ for all j. The coefficients of the polynomial $q(y)$ can be computed by using $O(n\log(n))$ ops. (One can evaluate $p(x)$ at the k-th roots of unity for $k > 2n$ and then interpolate to $q(y)$ by using $O(n\log(n))$ flops.)*

See [23] or [15] on the long history of the application of this transform to polynomial root-finding and see the seminal paper [58] on the first demonstration of the power of combining evaluation and interpolation.

MÖBIUS MAPS OF A LINE INTERVAL INTO A UNIT CIRCLE AND BACK

Theorem 2.4. *(i) The transforms $y = (x + 1/x)/2$ and $x = y \pm \sqrt{y^2 - 1}$ map the unit circle $C(0,1)$ into the real line interval $[-1,1] = \{y : \Im y = 0,\ -1 \le y \le 1\}$ and vice versa. (ii) Write $y = (x + 1/x)/2$ and $y_j = (x_j + 1/x_j)/2$, $j = 1,\dots,n$. Then $q(y) = p(x)p(1/x) = q_n\prod_{j=1}^{n}(y - y_j)$ (cf. [10, eq. (14)]). (iii) Given a polynomial $p(x)$ of (2.1) one can compute the coefficients of the polynomial $q(y) = p(x)p(1/x) = q_n\prod_{j=1}^{n}(y - y_j)$ by using $O(n\log(n))$ ops.*

Proof. Follow [10, Section 2]. Apply the algorithms of [38] to interpolate to the polynomial $q(y)$ from its values at the Chebyshev knots at the cost $O(n\log(n))$. □

Theorem 2.5. *Fix a complex $x = x^{(0)}$ and define the iteration*

$$x^{(h+1)} = (x^{(h)} - (x^{(h)})^{-1})/2 \text{ for } h = 0,1,\dots \tag{2.2}$$

If $x^{(0)}$ is real, then $x^{(h)}$ are real for all h. Otherwise $|x^{(h)} - \mathrm{sign}(x)\sqrt{-1}| \le \frac{2\tau^{2^h}}{1-\tau^{2^h}}$ for $\tau = |\frac{x-\mathrm{sign}(x)}{x+\mathrm{sign}(x)}|$ and $h = 0,1,\dots$

The theorem states simple extensions of the known estimates for the iteration $x^{(h+1)} = (x^{(h)} + (x^{(h)})^{-1})/2$ (cf. [10, page 500]).

Theorem 2.6. *Write $\gamma_i = |\lambda^{(i)} - \mathrm{sign}(\lambda^{(i)}|$ for $i = 0,1,\dots$. Assume (2.2) and $\gamma_0 \le 1/2$. Then $\gamma_i \le \frac{32}{113}(\frac{113}{128})^{3^i}$ for $i = 1,2,\dots$*

Proof. Complete the proof of [10, Proposition 4.1] by using the bound $\gamma_0 \le 1/2$. First verify that $\gamma_{i+1} = \gamma_i^3|3(\lambda^{(i)})^2 + 9\lambda^{(i)} + 8|/8$ and therefore $\gamma_{i+1} \le \frac{113}{32}\gamma_i^3$ for $i = 0,1,\dots$. Now the claimed bounds follow by induction on i for $\gamma_0 \le 1/2$. □

ROOT-FINDING WHERE ALL ROOTS ARE REAL.

In this special case the algorithms of [13], [11], [18], and [19] approximate all roots of a polynomial in optimal arithmetic and Boolean time (up to polylogarithmic factors).

Theorem 2.7. *The modified Laguerre algorithm of [18] and [19] converges to all roots of a polynomial $p(x)$ of (2.1) right form the start with superlinear convergence rate and uses $O(n)$ ops per iteration. Consequently the algorithm uses $O(\log(b))$ iteration loops, performing $\tilde{O}(n\log b)$ ops overall, in order to approximate within $\epsilon = 1/2^b$ all n roots. If $\log|x_i| = O(\log(n))$ for all roots x_i, then the iteration can be performed at the cost $\tilde{O}_B((b+\gamma)n)$. The alternative algorithms of [13] and [11] support the latter cost bound as well.*

APPROXIMATION OF THE ISOLATED ROOTS AND FACTORIZATION

The internal disc $D(X,r)$ of an annulus $A(X,R,r)$ is (R/r)-*isolated* and R/r is its *isolation ratio* if the polynomial $p(x)$ of (2.1) has no roots in the annulus.

Theorem 2.8. *(i) (Cf. [51, Corollary 4.5].) Suppose $\log(\rho) = O(\log(n))$ and the $5n^2$-isolated disc $D(0,\rho)$ contains a single simple root of a polynomial $p(x)$ of (2.1). Then Newton's iteration*

$$x^{(h+1)} = x^{(h)} - p(x^{(h)})/p'(x^{(h)}),\ h = 0,1,\dots \tag{2.3}$$

initiated at the center of the disc converges quadratically to this root right from the start, and so it approximates the root within $\epsilon = 1/2^b$ by using $\tilde{O}(n \log b)$ ops at the overall cost $\tilde{O}_B((b+\gamma)n)$. (ii) Both arithmetic and Boolean cost bounds can be extended to ϵ-approximation of k roots for any $k \le n$ if each of them is a single simple root in one of k given $5n^2$-isolated discs lying in the disc $D(0, \rho)$.

To prove part (ii) apply concurrently Newton's iteration (2.3) initialized at n the centers of the n isolated input discs and apply the Moenck–Borodin algorithm for simultaneous evaluation of the polynomials $p(x)$ and $p'(x)$ at the n points at each iteration. See, e.g., [39, Section 3.1] on this algorithm, [27, Theorem 3.7 and Algorithm 5.1]) or [47] on its Boolean cost estimates, and [42] and [43] on its efficient alternatives for numerical computations.

Remark 2.1. *Suppose we have computed some approximations $\tilde{r}_1, \dots, \tilde{r}_n$ to the root radii of a polynomial $p(x)$ of (2.1) (see Theorem 2.1 and see some alternative heuristic algorithms for root radii approximation in [2], [5], [12]). This defines $2n$ candidates $\pm\tilde{r}_1, \dots, \pm\tilde{r}_n$ for the approximation of the r real roots $x_1, \dots, x_r$. As we stated above, we can evaluate the polynomial at these $2n$ candidate points at a low arithmetic and Boolean cost, which would enable us to exclude a number of candidates. At the remaining candidate points we can initialize Newton's iteration. Then its single concurrent step or a few concurrent steps (all performed also at a low arithmetic and Boolean cost) should exclude the other extraneous candidates and would refine the remaining approximations to the real roots as long as they are sufficiently well isolated from the other roots of a polynomial, which is the case for a large class of input polynomials $p(x)$.*

At a low arithmetic and Boolean cost we can also split a polynomial into two factors if their roots are separated by an annulus that is not extremely thin.

Theorem 2.9. *(Cf. [55].) Suppose a polynomial $p(x)$ of (2.1) has k roots $z_1, \dots, z_k$ in the disc $D(0, r)$ and has $n-k$ roots outside the disc $D(0, R)$ for $0 < r < n/(n+1)$ and $R \ge 1 + 1/n$. Assume a positive tolerance $\epsilon = 1/2^b$ for $b \ge n$. Then one can compute two approximate factors $\tilde{f}$ and $\tilde{g}$ such that $||p - \tilde{f}\tilde{g}||_q \le \epsilon ||p||_q$ for $q = 1, 2$ or ∞, the polynomial $\tilde{f}$ has degree k and has k roots in the disc $D(0, 1)$, whereas the polynomial $\tilde{g}$ has degree $n-k$ and has $n-k$ roots outside that disc. The computation involves $\tilde{O}((b+n)n)$ Boolean operations.*

Remark 2.2. *The transform of Theorem 2.3 squares the isolation ratio of any disc $D(0, \rho)$. So it is sufficient to apply this transform $O(\log n)$ times (by using $O(n \log^2 n)$ ops or $\tilde{O}(bn)$ Boolean operations overall) to increase the isolation ratio of the disc from $1 + c/n^d$ for any pair of positive constants c and d to $1 + 1/n$ and even to $5n^2$. Having done this we can apply Theorems 2.8 and 2.9.*

3 Matrix Computations

FUNDAMENTALS

$M^T = (m_{ji})_{i,j=1}^{n,m}$ is the transpose of a matrix $M = (m_{ij})_{i,j=1}^{m,n}$. M^H is its Hermitian transpose. $I = I_n = (\mathbf{e}_1 \mid \mathbf{e}_2 \mid \dots \mid \mathbf{e}_n)$ is the $n \times n$ identity matrix whose columns are the n coordinate vectors $\mathbf{e}_1$, $\mathbf{e}_2, \dots, \mathbf{e}_n$. $\mathrm{diag}(b_j)_{j=1}^s = \mathrm{diag}(b_1, \dots, b_s)$ is the $s \times s$ diagonal matrix with the diagonal entries $b_1, \dots, b_s$.

A matrix Q is *unitary* if $Q^H Q = I$ or $QQ^H = I$. Let $(Q, R) = (Q(M), R(M))$ for an $m \times n$ matrix M of rank n denote a unique pair of unitary $m \times n$ matrix Q and upper triangular $n \times n$ matrix R such that $M = QR$ and all diagonal entries of the matrix R are positive [22, Theorem 5.2.2].

M^+ is the Moore–Penrose pseudo inverse of M [22, Section 5.5.4]. An $n \times m$ matrix $X = M^{(I)}$ is a left (resp. right) inverse of an $m \times n$ matrix M if $XM = I_n$ (resp. if $MY = I_m$). M^+ is an $M^{(I)}$ for a matrix M of full rank. $M^{(I)} = M^{-1}$ for a nonsingular matrix M.

$\mathcal{R}(M)$ is the range of a matrix M, that is the linear space generated by its columns. A matrix of full column rank is a *matrix basis* of its range.

EIGENSPACES

$\mathcal{S}$ is an *invariant subspace* or *eigenspace* of a square matrix M if $M\mathcal{S} = \{M\mathbf{v} : \mathbf{v} \in \mathcal{S}\} \subseteq \mathcal{S}$.

Theorem 3.1. *[56, Theorem 4.1.2], [60, Section 6.1], [61, Section 2.1]. Let $U \in \mathbb{C}^{n\times r}$ be a matrix basis for an eigenspace $\mathcal{U}$ of a matrix $M \in \mathbb{C}^{n\times n}$. Then the matrix $L = U^{(I)}MU$ is unique (that is independent of the choice of the left inverse $U^{(I)}$) and satisfies $MU = UL$ and consequently $\Lambda(L) \subseteq \Lambda(M)$.*

We call the above pair $\{L,\mathcal{U}\}$, as well as the pair $\{\lambda,\mathcal{U}\}$ if $L = \lambda I_n$, an *eigenpair* of a matrix M [56]. In the latter case $\det(\lambda I - M) = 0$, whereas $\mathcal{U}$ is the eigenspace associated with the *eigenvalue* λ and made up of its *eigenvectors*. $\Lambda(M)$ is the set of all eigenvalues of M, called its *spectrum*.

We readily verify the following results.

Theorem 3.2. *Suppose M is a square matrix, $\mathcal{U}$ is its eigenspace, and a rational function $f(M)$ is defined on the spectrum of the matrix M. Then $\Lambda(f(M)) = f(\Lambda(M))$, and $\mathcal{U}$ is an eigenspace of the matrix $f(M)$.*

RECOVERY OF THE EIGENVALUES

Suppose we have computed a matrix basis $U \in \mathbb{C}^{n\times r}$ for an eigenspace $\mathcal{U}$ of a matrix function $f(M)$ of an $n \times n$ matrix M. By virtue of Theorem 3.2 this is also a matrix basis of an eigenspace of the matrix M. We can compute a left inverse $U^{(I)}$ and then compute the eigenvalues of M associated with this eigenspace by solving the eigenproblem for the $r \times r$ matrix $L = U^{(I)}MU$ (cf. Theorem 3.1). Empirically the QR algorithm uses $O(r^3)$ ops at the latter stage.

If instead of (or in addition to) an eigenspace of the matrix function $f(M)$ we are given a reasonably close approximation $\tilde{x}$ to its eigenvalue, then we can proceed at a low computational cost as follows.

Algorithm 3.1. An eigenvalues of a matrix from the one of its function.

1. Compute a close approximation to an associated common eigenvector $\mathbf{u}$ of the matrices $f(M)$ and M by applying to the matrix $\tilde{x}I_n - f(M)$ one or a few loops of Inverse Power or Rayleigh Quotient iteration [22], [56].

2. Closely approximate an eigenvalue of M by the Rayleigh Quotient $\frac{\mathbf{u}^T M\mathbf{u}}{\mathbf{u}^T\mathbf{u}}$.

TOEPLITZ, CIRCULANT, SRFT, AND TOEPLITZ-LIKE MATRICES

An $n\times n$ Toeplitz matrix $T = (t_{i-j})_{i,j=1}^{n}$ is defined by the $2n-1$ entries of its first row and column. A circulant matrix $Z_1(\mathbf{t}) = (t_{i-j \mod n})_{i,j=1}^{n}$ is a special Toeplitz $n \times n$ matrix defined by its first column $\mathbf{t} = (t_i)_{i=0}^{n-1}$. They form an algebra generated by the *unit circulant* matrix $Z_1 = Z_1(\mathbf{e}_2)$.

Theorem 3.3. *(See [17].) Suppose $\omega = \exp(\frac{2\pi}{n}\sqrt{-1})$ denotes a primitive n-th root of unity, $\Omega = (\omega^{ij})_{i,j=0}^{n-1}$ denotes the $n \times n$ matrix of DFT (that is discrete Fourier transform), and $\mathbf{t} = (t_i)_{i=0}^{n-1}$ is a vector of dimension n. Then*

$$n\Omega^{-1} = \Omega^H = (\omega^{-ij})_{i,j=0}^{n-1} \quad \text{and} \quad Z_1(\mathbf{t}) = \Omega^{-1}\,\mathrm{diag}(\Omega\mathbf{t})\Omega.$$

SRFT (that is semisample random Fourier transform) $n \times l$ matrices for two fixed integers l and n, $1 < l < n$, are of the form $S = \sqrt{n/l}\ DFR$ where D is a random $n \times n$ diagonal matrix whose diagonal entries are i.i.d. variables uniformly distributed on the unit circle $C(0, 1)$, $F = \Omega$ is the DFT matrix, and R is a random $n \times l$ permutation matrix defined by random choice of l columns under the uniform probability distribution on the set of n columns of the identity matrix I_n [26, Section 11]. Theorem 3.3 implies the following fact.

Corollary 3.1. *Assume an $n \times l$ SRFT matrix $S = \sqrt{n/l}\ DFR$. Then $\Omega^{-1}S = Z_1(\mathbf{t})R$ for a circulant matrix $Z_1(\mathbf{t})$.*

A matrix M is Toeplitz-like and has structure of Toeplitz type if the rank d of its displacement $Z_1M - MZ_1$ is small (in context). Such a matrix can be expressed via short generators depending on $O(dn)$ parameters. Its multiplication by a vector is reduced to performing a small number of DFTs

and uses $O(dn\log(n))$ ops. Addition, multiplication, and inversion of Toeplitz-like matrices produce short generators for the output and use nearly linear arithmetic time, namely $O(d^b n\log^c(n))$ ops where $b=1$ and $c=0$ for addition and for multiplication by a scalar, $b=2$ and $c=1$ for pairwise multiplication and for the computation of a close approximation to the inverse, whereas $b=2$ and $c\le 2$ for the exact computation of the inverse (cf. [39], [33], [16], [62]).

COMPANION MATRICES

Hereafter $C_p=\begin{pmatrix} 0 & & & & -p_0/p_n\\ 1 & \ddots & & & -p_1/p_n\\ & \ddots & \ddots & & \vdots\\ & & \ddots & 0 & -p_{n-2}/p_n\\ & & & 1 & -p_{n-1}/p_n \end{pmatrix}$ denotes the companion matrix of the polynomial $p(x)$ of (2.1), such that $p(x)=c_{C_p}(x)=\det(xI_n-C_p)$ is the *characteristic polynomial* of C_p, that is, the set of the roots of $p(x)$ is the *spectrum* of C_p. This is a special Toeplitz-like matrix, and the following results hold.

Theorem 3.4. *(See [14] or [41].) The companion matrix $C_p\in\mathbb{C}^{n\times n}$ of a polynomial $p(x)$ of (2.1) generates an algebra $\mathcal{A}_p$. One needs $O(n)$ ops for addition, $O(n\log n)$ ops for multiplication and $O(n\log^2 n)$ ops for inversion in this algebra. Any matrix in this algebra is a Toeplitz-like matrix having displacement of rank at most 2, and so its product by a vector as well as a short generator for its product by an $n\times n$ Toeplitz or Toeplitz-like matrix can be computed by using $O(n\log n)$ ops.*

ITERATIONS IN THE FROBENIUS MATRIX ALGEBRA

For a polynomial $p(x)$ of (2.1) an a rational function $f(x)$ defined on the set $\{x_i\}_{i=1}^n$ of its roots, the rational matrix function $f(C_p)$ has the spectrum $\Lambda(f(C_p))=\{f(x_i)\}_{i=1}^n$, by virtue of Theorem 3.2. In particular the maps

$$C_p\to C_p^{-1},\ C_p\to aC_p+bI,\ C_p\to C_p^2,\ C_p\to 0.5(C_p+C_p^{-1}),\ \text{and}\ C_p\to 0.5(C_p-C_p^{-1})$$

induce the maps of the eigenvalues of the matrix C_p and thus induce the maps of the roots of the characteristic polynomial $p(x)$ given by the equations

$$y=1/x,\ y=ax+b,\ y=x^2,\ y=0.5(x+1/x),\ \text{and}\ y=0.5(x-1/x),$$

respectively. By using the reduction modulo $p(x)$ we define the five dual maps

$$y=(1/x)\mod p(x),\ y=ax+b\mod p(x),\ y=x^2\mod p(x),$$
$$y=0.5(x+1/x)\mod p(x),\ \text{and}\ y=0.5(x-1/x)\mod p(x)$$

where $y=y(x)$ denote polynomials. Apply the two latter maps recursively to define two iterations with polynomials modulo $p(x)$ as follows, $y_0=x$, $y_{h+1}=0.5(y_h+1/y_h)\mod p(x)$ (cf. (2.2)) and

$$y_0=x,\ y_{h+1}=0.5(y_h-1/y_h)\mod p(x),\ h=0,1,\dots. \tag{3.1}$$

More generally we can define the iteration

$$y_0=x,\ y_{h+1}=ay_h+b/y_h\mod p(x),\ h=0,1,\dots, \tag{3.2}$$

for any pair of scalars a and b. Here $y_h=y_h(x)$ are the polynomials in x, which are the characteristic polynomials of the matrices $M_0=C_p$, $M_{h+1}=0.5(M_h\pm M_h^{-1})$ and $M_0=C_p$, $M_{h+1}=aM_h+bM_h^{-1}$, $h=0,1,\dots$, respectively. The dual matrix representation of the iteration will help us at the stage where we recover the roots of a polynomial $p(x)$ of (2.1) from the roots of its image polynomial.

4 Real Root-finders

Theorem 2.5 implies that right from the start of iteration (2.2) the values $x^{(h)}$ converge to $\pm\sqrt{-1}$ exponentially fast unless the initial value $x^{(0)}$ is real, in which case all iterates $x^{(h)}$ are real. It follows that right from the start the values $y^{(h)} = (x^{(h)})^2 + 1$ converge to 0 exponentially fast unless $x^{(0)}$ is real, in which case all values $y^{(h)}$ are real and exceed 1. Write $q_h(y) = \prod_{j=1}^{n}(y - (x_j^{(h)})^2 - 1)$ for $h = 1, 2, \dots$ and $u_h(y) = \prod_{j=1}^{r}(y-(x_j^{(h)})^2-1)$. The roots of the polynomials $q_h(y)$ and $u_h(y)$ are the images of all roots and of the real roots of the polynomial $p(x)$ of (2.1), respectively, produced by the composition of the maps (2.2) and $y^{(h)} = (x^{(h)})^2 + 1$. Therefore $q_h(y) \approx y^{2s}u_h(y)$ for large integers h where every polynomial $u_h(y)$ has degree r and has exactly r real roots, all of them exceeding 1. Therefore, for sufficiently large integers h we can closely approximate the polynomial $u_h(y)$ simply by the sum of the $r + 1$ leading terms of the polynomial $q_h(y)$.

Alternatively we can apply the algorithms supporting Theorem 2.9 to approximate the factor $u_h(y)$ of the polynomial $q_h(y)$ with similar properties. In this case the cost of the approximation of the coefficients of the factor is higher than the cost of simply extracting these coefficients as the ones of $q_h(y)$, but is still low and, most important, we can approximate the factor for a smaller integer h, namely as soon as the $n - r$ roots converging to 0 move into the disc $D(0, n/(n + 1))$, whereas the other roots stay outside the unit disc $D(0, 1)$. We can detect this moment by applying the root radii algorithm that supports Theorem 2.1. Surely the same arguments apply where we use iteration (3.1) instead of (2.2), and we can extend them to the recursive application of the Möbius map of part (ii) of Theorem 2.4. In the latter case the iterates $x^{(h)}$ stay on the imaginary axis if they start there and otherwise converge to the points ± 1. So the values $y^{(h)} = (x^{(h)})^2 - 1$ either stay on the ray $\{x : x \le -1\}$ for all h or converge to 0 as $h \to \infty$.

Algorithm 4.1. Möbius iteration for real root-finding.

INPUT: *two integers n and r, $0 < r < n$, and the coefficients of a polynomial $p(x)$ of equation (2.1).*

OUTPUT: *approximations to the real roots $x_1, \dots, x_r$ of $p(x)$.*

INITIALIZATION: *Write $p_0(x) = p(-x\sqrt{-1})$.*

COMPUTATIONS:

1. *Recursively compute the polynomials $p_{h+1}(y) = p_h(x)p_h(1/x)$ for $y = (x + 1/x)/2$ and $h = 0, 1, \dots$ (cf. map (2.2)).*

2. *Periodically, at some selected Stages k, compute the polynomials*

$$t_k(y) = (-1)^n q_k(\sqrt{y+1})q_k(-\sqrt{y+1})$$

   where $q_k(z) = p_k(z)/\mathrm{lc}(p_k)$ (cf. Theorems 2.4 and 2.5), and apply to them the root radii algorithm supporting Theorem 2.1. If the algorithm shows that the disc $D(0, n/(n+1))$ contains $2s$ roots of the polynomial $t_k(x)$ (by extending Theorem 2.5 deduce that this must occurs if the integer k is sufficiently large), then apply the algorithm supporting Theorem 2.9 to approximate closely the factor $v_k(x)$ of the polynomial $t_k(x)$ that has r real roots on the ray $\{x : x \le -1\}$. Otherwise go back to the iteration of Stage 1.

3. *Apply one of the algorithms of [13], [11], [18], and [19] (cf. Theorem 2.7) to approximate the r roots $z_1, \dots, z_r$ of the polynomial $v_k(x)$.*

4. *Recover from them the r roots of the polynomial $p_0(x) = p(-x\sqrt{1})$ lying on the imaginary axis and then immediately obtain the r real roots of the polynomial $p(x)$.*

Suppose for a fixed (sufficiently large) integer k we have computed r common real roots of the polynomials $v_k(x)$ and $t_k(x)$.. Then we recover the r real roots of the polynomial $p(x)$ by extending the descending process from [35], also used in [40].

Namely, we first compute $2r$ candidates for r roots of the polynomial $q_k(y)$ lying on the imaginary axis and select exactly r of them on which the polynomial $q_k(y)$ vanishes. Similarly define from these

r roots $2r$ candidates for being the r roots of $p_{k-1}(x)$ lying on the imaginary axis. Then recursively descend down to the r roots of $p_0(x)$ lying on the imaginary axis. This process is not ambiguous because only r roots of $p_{k-1}(x)$ lie on that axis for each k.

Like lifting Stage 1, descending process involves $O(kn\log(n))$ ops, but for large integers k both stages would be prone to numerical stability problems if the condition numbers of the roots of the computed polynomials $p_k(y)$ grow exponentially as k grows large, to reach the level of the known upper estimates (cf. [10, Section 3]). Next, to avoid this deficiency we replace the polynomial iteration at Stages 1 and 2 by the dual matrix iteration and then recover the desired real eigenvalues of the matrix C_p by means of our recipes of Section 3.

Algorithm 4.2. Iterations with matrices for real root-finding.

INPUT AND OUTPUT *as in Algorithm 4.1, except that FAILURE can be output with a probability close to 0.*

COMPUTATIONS:

1. *Write $Y_0 = C_p$ and recursively compute the matrices*

$$Y_{h+1} = 0.5(Y_h - Y_h^{-1}) \text{ for } h = 0, 1, \dots. \tag{4.1}$$

   (For sufficiently large integers h the $2s$ eigenvalues of the matrix Y_h lie near the points $\pm\sqrt{-1}$, whereas the r other eigenvalues are real.)

2. *Fix a sufficiently large integer k and compute the matrix $Y_{k+1} = Y_k^2 + I_n$,*

3. *Apply the randomized algorithms of [26] to compute its numerical rank. If it exceeds r go back to Stage 1. Otherwise generate an $n \times l$ SRFT matrix $S = D\Omega R$ for a sufficiently large l of order $r\log(r)$. Compute the matrices $H = YS$ and $Q(H)$. (The matrices Y and $Y\Omega^{-1}$ have the same numerical ranks because the matrix Ω^{-1} is unitary up to scaling by $\sqrt{n}$.) Deduce from [26, Theorem 11.1] that with a probability close to 1 exactly $l - r$ columns of the matrix $Q(H)$ vanish or nearly vanish. If less than $l - r$ columns vanish or nearly vanish, then output FAILURE and stop.*

4. *Otherwise apply the recipes of Section 3 to approximate the r real eigenvalues of the matrix $M = C_p$, which are the roots of the polynomial $p(x)$.*

The arithmetic cost of the computations at Stages 1 and 2 is $O(kn\log(n))$ by virtue of Theorem 3.4, that is about the same as for Algorithm 4.1. Matrix multiplication of Y by $\Omega^{-1}S = (\Omega^{-1}D\Omega)R$ uses only $O(n\log(n))$ ops because the matrix $Y \in \mathcal{A}_p$ is Toeplitz-like, whereas $\Omega^{-1}D\Omega$ is a circulant matrix (see Corollary 3.1). The cost bounds are $O(nr^2)$ at Stage 4 and $O((k\log(n) + r^2)n)$ overall.

Remark 4.1. *At Stage 3 we compute numerical rank by applying multiplication by the same $n \times l$ SRFT matrix as we use for computing the matrix H, but for both tasks we can apply a standard Gaussian random $n \times r$ multiplier instead, which would imply success with superior probability estimates [44]. The cost of performing multiplication at Stage 3 would increase to $O(nr\log(n))$ ops, but this would only change the overall cost bound into $O(((k + r)\log(n) + r^2)n)$.*

Our next algorithm performs $O(nr^2)$ ops (rather than $O(nr\log(n))$) at Stage 4 and also extends the recipe of Algorithm 4.1 for choosing an integer k at Stage 2. Technically we accompany the computation of matrices of the algebra $\mathcal{A}_p$ involved into Algorithm 4.2 by the computation of their characteristic polynomials.

Algorithm 4.3. Iterations with polynomials and matrices for real root-finding.

INPUT AND OUTPUT *as in Algorithm 4.1.*

COMPUTATIONS:

1. *Write $y_0 = x$ and $Y_0 = C_p$ and compute the polynomials*

$$y_{h+1} = (y_h - y_h^{-1})/2 \mod p(x) \tag{4.2}$$

and the matrices $Y_{h+1} = 0.5(Y_h - Y_h^{-1})$ for $h = 0, 1, \ldots$ (cf. (4.1).

2. *Periodically compute the polynomials $t_k = y_k^2 + 1 \mod p(x)$ for selected integers k and approximate the root radii of these polynomials by applying the algorithm of Theorem 2.1. If the algorithm shows that the disc $D(0, \frac{n}{n+1})$ contains $2s$ roots of the polynomial t_k (by virtue of Theorem 2.5 this must be the case if integer k is sufficiently large), then apply the algorithm supporting Theorem 2.9 to approximate closely the factor v_k of the polynomial t_k, which has r real roots on the ray $\{x : x \geq 1\}$. Otherwise go back to Stage 1.*

3. *Apply one of the algorithms of [13], [11], [18], and [19] (cf. Theorem 2.7) to approximate the r roots $y_1, \ldots, y_r$ of the polynomial v_k, which are the real eigenvalues of the matrix $T_k = Y_k^2 + I$.*

4. *Recover the r real eigenvalues of the matrix C_p, which are the real roots of the polynomial $p(x)$ of (2.1), by first applying the Rayleigh Quotient Iteration to the matrices $T_k - y_j I_n$ for $j = 1, \ldots, r$ to compute the eigenvectors of the matrix C_p associated with its real eigenvalues and then readily recover these eigenvalues as the Rayleigh Quotients (cf. Algorithm 3.1).*

Remark 4.2. *Stage 1 of the algorithm combines square root iteration (4.2) with polynomials and matrix sign iteration (4.1). Algorithm 4.2 uses just iteration (4.1), but we can devise an algorithm using only iteration (4.2). We would proceed similarly to Algorithm 4.1, except that we would write $p_0(x) = p(x)$, rather than $p_0(x) = p(-x\sqrt{-1})$, would use iteration (4.2) at Stage 1, and would replace the expression $t_k(y) = (-1)^n q_k(\sqrt{y+1}) q_k(-\sqrt{y+1})$ with $t_k(y) = (-1)^n q_k(\sqrt{y-1}) q_k(-\sqrt{y-1})$ at Stage 2. As in Algorithm 4.1 we would use the descending process rather than the recipes of Section 3 at the recovery stage.*

5 Discussion

We have combined the square root iteration with polynomials and the matrix sign iteration in the Frobenius algebra generated by the companion matrix of a given polynomial. This turned out to produce efficient algorithms for the approximation of real roots of a polynomial. The iterations benefited from employing the duality between operations with matrices and polynomials. This synergistic idea should have more applications to root-finding and other computations with matrices and polynomials (cf. [34], [9], [39]). In particular it can help strengthen our present algorithms by various modifications, which we omitted in our initial study, presented in this paper.

For example, the following customary scaling of the matrix sign iteration dramatically accelerates its global convergence and improves its numerical behavior (cf. [25]),

$$Y_{h+1} = 0.5(\nu_h Y_h - (\nu_h Y_h)^{-1}) \text{ for } \nu_h^2 = ||Y_h^{-1}||_2 / ||Y_h||_2 \text{ and } h = 0, 1, \ldots.$$

We can extend this scaling to the dual polynomial iteration as follows,

$$y^{(h+1)} = 0.5(\nu_h y^{(h)} - (\nu_h y^{(h)})^{-1}) \mod p(x)$$

where we can use the same scalars ν_h, defined by the matrices Y_h, or can try various expressions in terms of the norms $||y^{(h)} \mod p(x)||$ and $||(y^{(h)})^{-1} \mod p(x)||$. Furthermore we can propose and test various tentative expressions for our algorithm of Remark 4.2 as well as Algorithms 4.2 and 4.3.

We can benefit further from extending some variations of the matrix sign iteration such as the Padé and Halley iterations or the ones of [25, equations (6.17)–(6.20)]. Some variations avoid matrix inversions or converge faster when the images of the nonreal eigenvalues of the matrix C_p are moved into certain large neighborhoods of the points of attraction, in our case $\pm\sqrt{-1}$. For example, here is our simple extension of the [0/2] Padé iteration from [25],

$$Y_{h+1} = -0.125(3Y_h^5 + 10Y_h^3 + 15Y_h) \text{ for } h = 0, 1, \ldots.$$

It confines the computations to real values in the case of real inputs, converges cubically if initiated in any of two large discs centered at the points $\pm\sqrt{-1}$, and involves no matrix inversions.

Many other amendments should be incorporated to improve practical performance of our algorithms. In particular our iteration may involve absolutely large real roots of some auxiliary polynomials and absolutely large real eigenvalues of some auxiliary matrices, but we plan to apply real shifts and scaling of the variable, defined deterministically or randomly, to avoid this undesired growth. See [49], [50] on these and other recipes for numerical stabilization of the iteration, on its application in the Frobenius algebra to approximation of the absolutely largest roots of a polynomial and on the matrix version of the Weyl Quad Tree construction for root-finding.

We also plan to explore the extensions of our techniques to splitting out some nonreal roots and eigenvalues, lying far from the real axis. In this case we would deflate the input polynomial or its companion matrix to decrease the problem size.

In another modification we would seek r_+ approximations to all real and ϵ-real roots, the latter lying within a fixed tolerance ϵ from the real axis. Having these r_+ approximations available, we would refine them and select the r real roots. The overall computational cost is low if $r_+ \ll n$.

Finally here is a promising extension of our real root-finding techniques where we employ repeated squaring of the roots instead of mapping them into their square roots.

Algorithm 5.1. Real root-finding by means of repeated squaring.

For an input polynomial $p(x)$ of (2.1) apply the following steps.

1. Compute the polynomial $y = (x+\sqrt{-1})(x-\sqrt{-1})^{-1} \mod p(x)$. This Cayley map moves the real axis (with real roots of $p(x)$) to the unit circle $C(0,1)$.

2. Write $y^{(0)} = y$, choose a sufficiently large integer k and apply the k squaring steps $y^{(h+1)} = (y^h)^2 \mod p(x)$, for $h = 1, \dots, k-1$. These steps keep the images of the real roots of $p(x)$ on the circle $C(0,1)$ for any k and send the images of s other roots of $p(x)$ toward 0 and the s remaining roots toward ∞ where $2s = n - r$.

3. Note that for a sufficiently large integer k the polynomial $y^{(k)}$ approximates the polynomial $x^s u_k(x)$ where the polynomial $u_k(x) = \sum_{i=0}^{r} u_i x^i$ has all its roots lying on the unit circle $C(0,1)$. Extract the approximation to this polynomial $u_k(x)$ from the coefficients of the polynomial $y^{(k)}$.

4. Compute the polynomial $v_k(x) = \sqrt{-1}(u_k(x)+1)(u_k(x)-1)^{-1} \mod p(x)$. This inverse Cayley map moves the images of the real roots of the polynomial $p(x)$ from the unit circle $C(0,1)$ back to the real line.

6. Apply one of the algorithms of [13], [11], [18], and [19] (cf. Theorem 2.7) to approximate the r real roots $z_1, \dots, z_r$ of the polynomial $v_k(x)$.

7. Apply the Cayley map $w_j = (z_j+\sqrt{-1})(z_j-\sqrt{-1})^{-1}$ for $j = 1, \dots, r$ to approximate the r roots $w_1, \dots, w_r$ of the polynomials $u_k(x)$ and $y_k(x) = x^s u_k(x)$ lying on the unit circle $C(0,1)$.

8. Apply the descending process (similar to the ones of [35]–[37], [40], and of our Algorithm 4.1) to approximate the r roots $x_1^{(h)}, \dots, x_r^{(h)}$ of the polynomials $y_h(x)$ lying on the unit circle $C(0,1)$ for $h = k-1, \dots, 0$.

10. Apply the inverse Cayley map to approximate the r real roots $x_j = (x_j^{(0)}+\sqrt{-1})(x_j^{(0)}-\sqrt{-1})^{-1}$ of the polynomials $p(x)$.

The claimed behavior of the roots of the polynomials involved in the algorithms follows from Theorem 3.2 because these polynomials are the characteristic polynomials of the matrix functions defined by the dual recursive iterative processes with the associate matrices of the algebra $\mathcal{A}_p$. The latter processes themselves do not work numerically because repeated squaring of the matrix $Y = (C_p + I_n\sqrt{-1})(C_p - I_n\sqrt{-1})^{-1}$ (dual to the polynomial y) turns it very soon into a matrix having numerical rank s. Indeed the domination of the s absolutely largest eigenvalues of the matrix Y becomes very strong in the process of repeated squaring. Nevertheless we can use these matrices indirectly, for the analysis of our computations with their characteristic polynomials.

Remark 5.1. *At Stage 3 we can apply twice the algorithm supporting Theorem 2.9 as an alternative means of the approximation of the factor of degree r that has the roots being the images of the real roots of $p(x)$. This alternative costs more than the original recipe of just copying the $r+1$ coefficients of the polynomial $y_k(x)$ at Stage 3, but is still performed at low arithmetic and Boolean cost and works*

for smaller integers k, that is as soon as the discs $D(0, \frac{n}{n+1})$ and $D(0,1)$ become $(1+1/n)$-isolated with respect to the polynomial $y^{(k)}(x)$.

Acknowledgement: Our research has been supported by the NSF Grant CCF 1116736.

References

[1] J. L. Aurentz, R Vandebril, D. S. Watkins, Fast computation of the zeros of a polynomial via factorization of the companion matrix *SIAM Journal on Scientific Computing*, **35 1**, 255-269, 2013.

[2] D. Bini, Numerical Computation of Polynomial Zeros by Means of Aberth's Method, *Numerical Algorithms* **13**, 179–200, 1996.

[3] D. A. Bini, P. Boito, Y. Eidelman, L. Gemignani, I. Gohberg, A Fast Implicit QR Algorithm for Companion Matrices, *Linear Algebra and Its Applications*, **432**, 2006–2031, 2010.

[4] D. A. Bini, F. Daddi, and L. Gemignani, On the Shifted QR Iteration Applied to Companion Matrices, *Electronic Transactions on Numerical Analysis (ETNA)*, **18**, 137–152, 2004.

[5] D. A. Bini, G. Fiorentino, Design, Analysis, and Implementation of a Multiprecision Polynomial Rootfinder, *Numerical Algorithms*, **23**, 127–173, 2000.

[6] D. A. Bini, L. Gemignani, V. Y. Pan, Inverse Power and Durand/Kerner Iteration for Univariate Polynomial Root-finding, *Computers and Mathematics (with Applications)*, **47**, **2/3**, 447–459, 2004. (Also Technical Report TR 2002 020, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, 2002.)

[7] D. A. Bini, L. Gemignani, V. Y. Pan, Improved Initialization of the Accelerated and Robust QR-like Polynomial Root-finding, *Electronic Transactions on Numerical Analysis*, **17**, 195–205, 2004.

[8] D. A. Bini, L. Gemignani, V. Y. Pan, Fast and Stable QR Eigenvalue Algorithms for Generalized Companion Matrices and Secular Equation, *Numerische Math.*, **3**, 373–408, 2005. (Also Technical Report 1470, *Department of Math, University of Pisa*, Pisa, Italy, July 2003.)

[9] D. Bini, V. Y. Pan, *Polynomial and Matrix Computations, Volume 1: Fundamental Algorithms*, Birkhäuser, Boston, 1994.

[10] D. Bini, V. Y. Pan, Graeffe's, Chebyshev, and Cardinal's Processes for Splitting a Polynomial into Factors, *J. Complexity*, **12**, 492–511, 1996.

[11] D. Bini and V. Y. Pan, Computing Matrix Eigenvalues and Polynomial Zeros Where the Output Is Real, *SIAM J. on Computing* **27, 4**, 1099–1115, 1998 (Also in Proc. of *SODA'1991*.)

[12] D. A. Bini. L. Robol, Solving secular and polynomial equations: A multiprecision algorithm, *J. Computational and Applied Mathematics*, in press. Available at http://dx.doi.org/10.1016/j.cam.2013.04.037

[13] M. Ben-Or, P. Tiwari, Simple algorithms for approximating all roots of a polynomial with real roots, *Journal of Complexity*, **6, 4**, 417–442, 1990.

[14] J. P. Cardinal, On Two Iterative Methods for Approximating the Roots of a Polynomial, *Lectures in Applied Mathematics*, **32** (*Proceedings of AMS-SIAM Summer Seminar: Mathematics of Numerical Analysis: Real Number Algorithms* (J. Renegar, M. Shub, and S. Smale, editors), Park City, Utah, 1995), 165–188, American Mathematical Society, Providence, Rhode Island, 1996.

[15] F. Cajori, *A History of Mathematics, 5/E*, AMS Chelsea Publ., Providence, Rhode Island, 1999.

[16] S. Chandrasekaran, M. Gu, X. Sun, J. Xia, J. Zhu, A Superfast Algorithm for Toeplitz Systems of Linear Equations, *SIAM J. Matrix Anal. Appl.*, **29**, 1247–1266, 2007.

[17] R. E. Cline, R. J. Plemmons, and G. Worm, Generalized Inverses of Certain Toeplitz Matrices, *Linear Algebra and Its Applications,* **8**, 25–33, 1974.

[18] Q. Du, M. Jin, T. Y. Li, Z. Zeng, Quasi-Laguerre Iteration in Solving Symmetric Tridiagonal Eigenvalue Problems, *SIAM J. Sci. Computing*, **17**, **6**, 1347–1368, 1996.

[19] Q. Du, M. Jin, T. Y. Li, Z. Zeng, The Quasi-Laguerre Iteration, *Math. of Computation*, **66**, **217**, 345–361, 1997.

[20] I. Z. Emiris, A. Galligo, E. P. Tsigaridas, Random polynomials and expected complexity of bisection methods for real solving, *Proc. 35th Ann. Intern. Symp. on Symbolic and Algebraic Computation (ISSAC'2010)* (S. Watt, editor), 235–242, ACM Press, New York, 2010.

[21] S. Fortune, An Iterated Eigenvalue Algorithm for Approximating Roots of Univariate Polynomials, *J. of Symbolic Computation*, **33**, **5**, 627–646, 2002. (Proc. version in *Proc. Intern. Symp. on Symbolic and Algebraic Computation (ISSAC'01)*, 121–128, ACM Press, New York, 2001.)

[22] G. H. Golub, C. F. Van Loan, *Matrix Computations* (third edition), The Johns Hopkins University Press, Baltimore, Maryland, 1996.

[23] A. S. Householder, Dandelin, Lobachevskii, or Graeffe, *American Mathematical Monthly* **66**, 464–466, 1959.

[24] A. S. Householder, Generalization of an Algorithm by Sebastiao e Silva, *Numerische Math.*, **16**, 375–382, 1971.

[25] N. J. Higham, *Functions of Matrices: Theory and Computations*, SIAM, Philadelphia, 2008.

[26] N. Halko, P. G. Martinsson, J. A. Tropp, Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions, *SIAM Review*, **53, 2**, 217–288, 2011.

[27] P. Kirrinnis, Polynomial Factorization and Partial Fraction Decomposition by Simultaneous Newton's Iteration, *J. of Complexity*, **14**, 378–444, 1998.

[28] A. Mantzaflaris, B. Mourrain, E. P. Tsigaridas, On Continued Fraction Expansion of Real Roots of Polynomial Systems, Complexity and Condition Numbers, *Theoretical Computer Science*, **412, 22**, 2312–2330, 2011.

[29] McNamee, J. M. (1993), Bibliography on Roots of Polynomials, *J. Computational and Applied Mathematics*, **47**, 391–394.

[30] McNamee, J. M. (2002), A 2002 Update of the Supplementary Bibliography on Roots of Polynomials, *J. of Computational and Applied Math.* **142**, 433–434

[31] J.M. McNamee, *Numerical Methods for Roots of Polynomials (Part 1)*, Elsevier, Amsterdam, 2007.

[32] J. M. McNamee, V. Y. Pan, Numerical Methods for Roots of Polynomials, Part 2 (XXII + 718 pages), Elsevier, 2013.

[33] P. G. Martinsson, V. Rokhlin, M. Tygert, A Fast Algorithm for the Inversion of Toeplitz Matrices, *Comput. Math. Appl.*, **50**, 741–752, 2005.

[34] V. Y. Pan, Complexity of Computations with Matrices and Polynomials, *SIAM Review*, **34**, **2**, 225–262, 1992.

[35] V. Y. Pan, Optimal (up to Polylog Factors) Sequential and Parallel Algorithms for Approximating Complex Polynomial Zeros, *Proc. 27th Ann. ACM Symp. on Theory of Computing*, 741–750, ACM Press, New York, 1995.

[36] V. Y. Pan, Optimal and Nearly Optimal Algorithms for Approximating Polynomial Zeros, *Computers and Math. (with Applications)* **31, 12**, 97–138, 1996.

[37] V. Y. Pan, Solving a Polynomial Equation: Some History and Recent Progress, *SIAM Review*, **39, 2**, 187–220, 1997.

[38] V. Y. Pan, New Fast Algorithms for Polynomial Interpolation and Evaluation on the Chebyshev Node Set, *Computers and Math. (with Applications)*, **35, 3**, 125-129, 1998.

[39] V. Y. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*, Birkhäuser, Boston, and Springer-Verlag, New York, 2001.

[40] V. Y. Pan, Univariate Polynomials: Nearly Optimal Algorithms for Factorization and Rootfinding, *Journal of Symbolic Computations*, **33**, **5**, 701–733, 2002. Proc. version in *ISSAC'2001*, pages 253–267, ACM Press, New York, 2001.

[41] V. Y. Pan, Amended DSeSC Power Method for Polynomial Root-finding, *Computers and Math. (with Applications)*, **49**, **9**–**10**, 1515–1524, 2005.

[42] V. Y. Pan, Fast Approximate Computations with Cauchy Matrices, Polynomials and Rational Functions, *Proc. of the Ninth International Computer Science Symposium in Russia (CSR'2014),* (E. A. Hirsch et al., editors), Moscow, Russia, June 2014, Lecture Notes in Computer Science (LNCS), **8476**, pp. 287–300 Springer International Publishing, Switzerland, 2014.

[43] V. Y. Pan, Transformations of Matrix Structures Work Again, accepted by *Linear Algebra and Its Applications,* 2014. Available at arxiv:1311.3729[math.NA]

[44] V. Y. Pan, G. Qian, X. Yan, Supporting GENP and Low-rank Approximation with Random Multipliers, preprint, 2014.

[45] Pan, V. Y., Qian, G., Zheng, A., Chen, Z. (2011a), Matrix computations and polynomial root-finding with preprocessing, *Linear Algebra and Its Applications*, **434**, 854–879.

[46] Pan, V. Y., Qian, G., Zheng, A. (2012a), Real and Complex Polynomial Root-finding via Eigen-Solving and Randomization, *Proceedings of Workshop on Computer Algebra in Scientific Computing (CASC 2012)*, (V. P. Gerdt et al. editors), *Lecture Notes in Computer Science*, **7442**, 283–293, Springer, Heidelberg.

[47] V. Y. Pan, E. P. Tsigaridas, Nearly Optimal Computations with Structured Matrices, April 18, 2014. Available at arXiv:1404.4768 [math.NA], 2014.

[48] V. Y. Pan, A. Zheng, New Progress in Real and Complex Polynomial Root-Finding, *Computers Math. Applics.* **61**, 1305–1334.

[49] V. Y. Pan, A. Zheng, New Structured Matrix Methods for Real and Complex Polynomial Root-finding" (by Victor Y. Pan, Ai-Long Zheng), Nov. 23, 2013. Available at arXiv 1311.6077 math.NA

[50] V. Y. Pan, A. Zheng, New Algoirthms in the Frobenius Matrix Algebra, Technical Report TR 2014006, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, 2014.

Available at http://www.cs.gc.cuny.edu/tr/techreport.php?id=469

[51] J. Renegar, On the worst-case arithmetic complexity of approximating zeros of polynomials, *Journal of Complexity*, **3, 2**, 90–113, 1987.

[52] F. Rouillier, On solving systems of bivariate polynomials, *Mathematical Software–ICMS 2010*, **6327**, 100–104, Springer, 2010.

[53] J. Sebastiao e Silva, Sur une méthode d'approximation semblable a celle de Graeffe, *Portugal Math.*, **2**, 271–279, 1941.

[54] G. W. Stewart, On the Convergence of Sebastiao E Silva's Method for Finding a Zero of a Polynomial, *SIAM Review*, **12**, 458–460, 1970.

[55] A. Schönhage, The Fundamental Theorem of Algebra in Terms of Computational Complexity, *Mathematics Department, University of Tübingen*, Germany, 1982.

[56] G. W. Stewart, *Matrix Algorithms, Vol II: Eigensystems*, SIAM, Philadelphia, 1998 and 2001.

[57] M. Sagraloff, When Newton meets Descartes: A simple and fast algorithm to isolate the real roots of a polynomial, *Proc. 37th Ann. Int'l Symp. on Symbolic and Algebraic Comp. (ISSAC '2012)*, 297–304, ACM Press, New York, 2012.

[58] A. L. Toom, The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers, *Soviet Mathematics Doklady*, **3**, 714–716, 1963.

[59] M. Van Barel, R. Vandebril, P. Van Dooren, K. Frederix, Implicit Double Shift QR-algorithm for Companion Matrices, *Numerische Mathematik* **116**, **2**, 177–212, 2010.

[60] D. S. Watkins, *Fundamentals of Matrix Computations*, Wiley, New York, 2002 (second edition).

[61] D. S. Watkins, *The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods*, SIAM, Philadelphia, PA, 2007.

[62] J. Xia, Y. Xi, M. Gu, A superfast structured solver for Toeplitz linear systems via randomized sampling, *SIMAX*, **33**, 837–858, 2012.